User-Centric Development

SPC-94061-CMC

Version 01.00.04
December 1994

Christine Haapala
James Kirby, Jr.

Producedby the
SOFTWARE PRODUCTIVITY CONSORTIUM SERVICES CORPORATION
under contract to the
VIRGINIA CENTER OF EXCELLENCE
FOR SOFTWARE REUSE AND TECHNOLOGY TRANSFER

SPC Building
2214 Rock Hill Road
Herndon, Virginia 22070

User-Centric Development

SPC-94061-CMC
Version 01.00.04

December 1994

CONTENTS

FIGURES

TABLES

ACKNOWLEDGMENTS

Thanks go to Steve Cross and Craig Wier of Advanced Research Projects Agency who sponsored and guided this study; to Neil Burkhard, Jim Marple, and Steve Wartik whose thoughtful reviews improved this report; to Bobbie Troy who edited the report; to Debbie Morgan and Deborah Tipeni who word processed it.

Special thanks go to the 17 participants who contributed their time to this study.

This page intentionally left blank.

EXECUTIVE SUMMARY

This technical report describes a case study of at least three software development projects that have employed user-centric software development (UCD). Characteristics of UCD are:

- The software development team includes some of those who will use the software. The user is an equal participant with the developers in making development decisions.
- The software development team develops the software using an iterative enhancement approach. A series of versions of the software is developed and delivered with feedback from users of earlier versions of the software driving development of later versions.

Results of the case study were obtained by posing a set of questions to those participating in UCD as developers, users, or managers. The questions, captured as a set of assertions, were used to test a theory of UCD. The results describe how a particular software development paradigm was practiced, the advantages and disadvantages of using the paradigm, technology transfer issues, and opportunities for research and technology advancements.

The intended audience for this technical report consists of:

- Technologists interested in evaluating, comparing, improving, and supporting software development methods and processes
- Practitioners responsible for selecting the software development methods and processes that software development projects will employ

This page intentionally left blank.

1. INTRODUCTION

1.1 PURPOSE

User-centric development (UCD) is a software development process characterized by a well-balanced team, consisting of users, customers, and developers, developing a system using iterative enhancement. This paper reports on a broad case study of a number of projects that have successfully practiced UCD. The case study is broad in the sense that it touches on a number of issues relevant to UCD, for example, software development methods and tools, management, the role of teams, building teams, and technology transfer.

This report will help readers understand how UCD is practiced, some of the advantages and disadvantages of using it, issues in transferring it into an organization, and opportunities for research and technology advancements.

1.2 AUDIENCE

The audience for this report consists of those who are:

- Interested in evaluating, comparing, improving, and supporting software development methods and processes
- Responsible for selecting the software development methods and processes that software development projects will employ

1.3 ORGANIZATION

This report is organized in the following way:

- Section 2 discusses UCD and the literature related to it.
- Section 3 characterizes the projects practicing UCD that the Software Productivity Consortium (Consortium) interviewed for the study.
- Section 4 describes how this case study was conducted.
- Section 5 describes the propositions about UCD that drove the interviews.
- Section 6 compares notes from the interviews to the propositions described in Section 5.

- Section 7 discusses conclusions of the study and proposes future work based on what was learned in this study.
- Appendix A presents the questions that drove the interviews the Consortium conducted.

1.4 TYPOGRAPHIC CONVENTIONS

This report uses the following typographic conventions:

Serif font..General presentation of information.

Italicized serif font...............................Publication titles.

2. USER-CENTRIC DEVELOPMENT

2.1 CHARACTERISTICS OF USER-CENTRIC DEVELOPMENT

UCD is a software development process characterized by a well-balanced team, which focuses on developing a system with a user addressing a significant challenge. Most importantly, the user is intimately involved throughout the full life cycle of UCD. UCD can best be described by characteristics in the areas of team, process, and challenge.

2.1.1 THE ORCHESTRATED PROJECT TEAM

According to Constantine and Lockwood (1993), a project team:

- Includes the users of the software, as well as developers and customers (i.e., a program office that finances the project for the user)
- Is empowered (has authority and responsibility) to respond quickly, in particular, to the user's requests and feedback
- Is distinguished by the presence or the emergence of a leader
- Is composed of members with well-differentiated and specific team roles

Having users on the software development team shortens the time that software developers need to discover user requirements, to learn the application domain, and to produce products that satisfy users. In turn, the users must learn about technology from the system experts in order to understand what computer technology can do for them. The users cannot describe what they really do because they have internalized their efforts. Not until they are questioned in the context of their work, are they able to reflect and converse about the specifics of their work. Furthermore, being in the user's environment provides a real-life experience of the activities a system must support (Clement and Van den Besselaar 1993). By putting systems designers in the workspace of the users, then design decisions can be made that accurately address the user's current needs.

Because "increased competition in international markets is pushing companies to adopt new structures that reduce middle management and move decision making to lower levels of the hierarchy," (Clement 1994) the UCD's empowered team is positioned to meet this economic reality and, thus, respond quickly to the user. The empowered team functions "with less supervision and assumes wider task responsibility" (Clement 1994).

The presence of a leader on the UCD team, most notably in the role of player-coach, gives the team more than the typical management guidance (Haapala, Hess, and Shore 1988). A UCD leader must "challenge the process, inspire a shared vision, enable others to act, model the way, and encourage the heart" (Funk 1992).

Each of the UCD's team members has a specific contribution and is an active participant. The UCD team acts cooperatively and collaboratively to achieve a common purpose (Funk 1992).

2.1.2 The Iterative Enhancement Process

According to Basili and Turner (1975), the iterative enhancement process:

- Is an incremental, evolutionary growth in project maturity by responding to the feedback from the users of the software
- Drives the development of a series of versions of the software in a short, repetitive cycle of preplanned functionality

It is a well-known fact that the user often does not know what he wants or, at a minimum, is unable to articulate it (DeBellis and Haapala 1994). Additionally, "for every application beyond the trivial, users' needs are constantly evolving ... aiming at a moving target" (Davis, Bersoff, and Comer 1988). Using the technique of incremental development, "the process of constructing a partial implementation and slowly adding increased functionality" (Davis, Bersoff, and Comer 1988) increases the chance that through these frequent insertions, the product will hit the moving target.

As early as the mid 70s, the iterative approach was identified as a "practical approach ... a simple initial implementation of a subset of the problem ... iteratively enhance existing versions until the full system is implemented" (Basili and Turner 1975). While these steps might appear to advance slowly through incremental improvements, in actuality, quantum leaps in system development and cognitive leaps in requirements understanding, both from the user and the system developer, often occur. Delivering a series of versions that are responsive to the users ensures their continuing interest and commitment.

2.1.3 The Significant Challenge

- In the long term, UCD centers around a significant and compelling vision of the user's future needs.
- In the short term, UCD focuses on delivering a limited version very early and subsequently building, through the requirements enhancement process, a continually maturing system.

Vision through a compelling picture of the future is a significant factor in UCD. UCD demands a long-term future orientation that is "looking beyond the horizon of the present" (Funk 1992).

Short-term goals mapped to the vision are "sharp milestones." The short-term goals characteristically are achieved through iterations or interim system builds occurring on a frequency of less than 8 to 10 weeks. While the frequent deliveries may be considered a burden or distraction by traditional software developers, these goals or milestones "are, in fact, a service to the team... a fuzzy milestone is the harder burden to live with ... a millstone that grinds down morale" (Brooks 1972).

Section 2.2 describes an application of UCD in developing an advanced analytical intelligence system.

2.2 A USER-CENTRIC DEVELOPMENT EXAMPLE

Over the last decade, the Joint National Intelligence Development Staff (JNIDS) has successfully applied a user-centric, iterative development process to develop advanced analytic intelligence

systems (Haapala, Hess, and Shore 1988; Haapala 1992). The process includes the dimensions of an empowered, energetic project team; continuous user involvement; and a rapid, systematic implementation of user requirements through iterative enhancement.

2.2.1 THE TEAM

Each JNIDS project has a core project team that is a diverse partnership involving three separate organizational groups: JNIDS, user(s), and the system developer(s). JNIDS, as the government program office, provides the implementation budget and management direction; the user, as the ultimate recipient of the final system, provides the functional vision and requirements direction; and the development contractor, as the implementation arm of the team, transforms the functional vision into an operating system. According to one JNIDS development partner (National Information Display Lab 1992), "the key to the JNIDS' approach is the continual involvement of Government users in a rapid, iterative design process which provides users with the operational system."

Since JNIDS does not assume that the domain knowledge of the user could be captured on paper or purchased through surrogate users in the form of contractors, the user was embraced as an integral, crucial part of the core team. Drucker recognized the nonexistence of a "universally gifted" man (Drucker 1966). Similarly, JNIDS recognized the nonexistence of a "universally gifted" organization and executed their projects through a virtual organization of teams and partnerships (Haapala 1993; JNIDS 1993). This virtual organization was always user focused and "replaces the functional hierarchies and matrix structures with flexible, empowered, inspired, integrated business teams that have few levels of management and function more like a community" (Luftman, Lewis, and Oldach 1993).

2.2.2 THE PROCESS

The key development activities in the waterfall software development process, in large part, inspired the JNIDS iterative development process. The activities of requirements, design, code, integration, and system implementation are preserved and remain necessary, distinct activities in the JNIDS process in Figure 1. Additionally, because the core project team was such an important aspect, another dimension of the software development process was explicitly identified: team orchestration. Team orchestration represents the effort and time involved to move the team through the team-building, formative stages of "forming, storming, norming and performing."

Rather than performing these development activities in series, only once, with the output from one being input to the other as in Figure 5, they occur in parallel multiple times, as in Figure 2. Viable results, in the form of operational prototypes, occur every 8 weeks rather than in years or decades.

The JNIDS iterative development process as presented in Figure 3 is a phased, parallel process that instantiates a user-focused vision from the initial requirements gathering stage through simultaneous implementation of a partial solution to the final delivery of the operational system.

The goal of Phase 0 is to identify requirements through a functional analysis of the user's high-level needs, develop a high-level Statement of Work, issue a competitive Request for Proposals, and award a contract. Following award of contract, the goal of Phase 1 is to field a first-cut operational prototype in the user's environment, within 6 months. Phases 2, 3, and 4 are each year long. The goals for Phase 2 and 3 are to deliver, every 8 weeks, enhanced versions of the operational prototype based on the user's feedback during operational use of the system. The primary difference between Phase 2 and Phase 3 is that Phase 2 concentrates on the breadth of the system functionality, while Phase 3

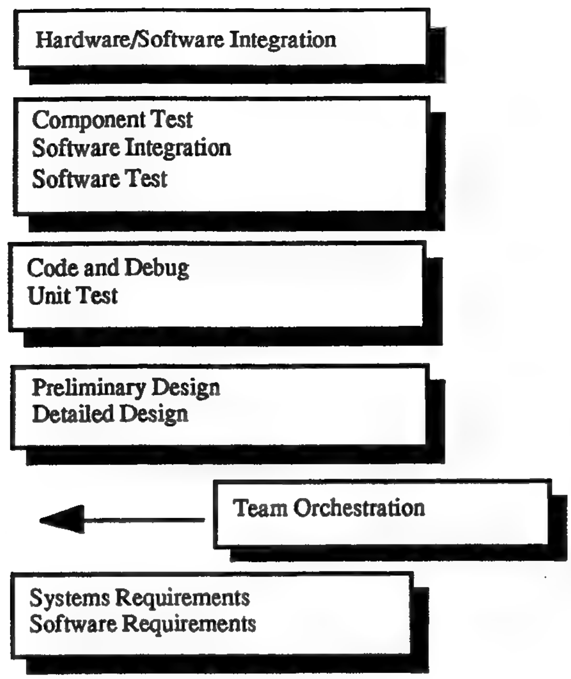

Figure 1. JNIDS Process Activities

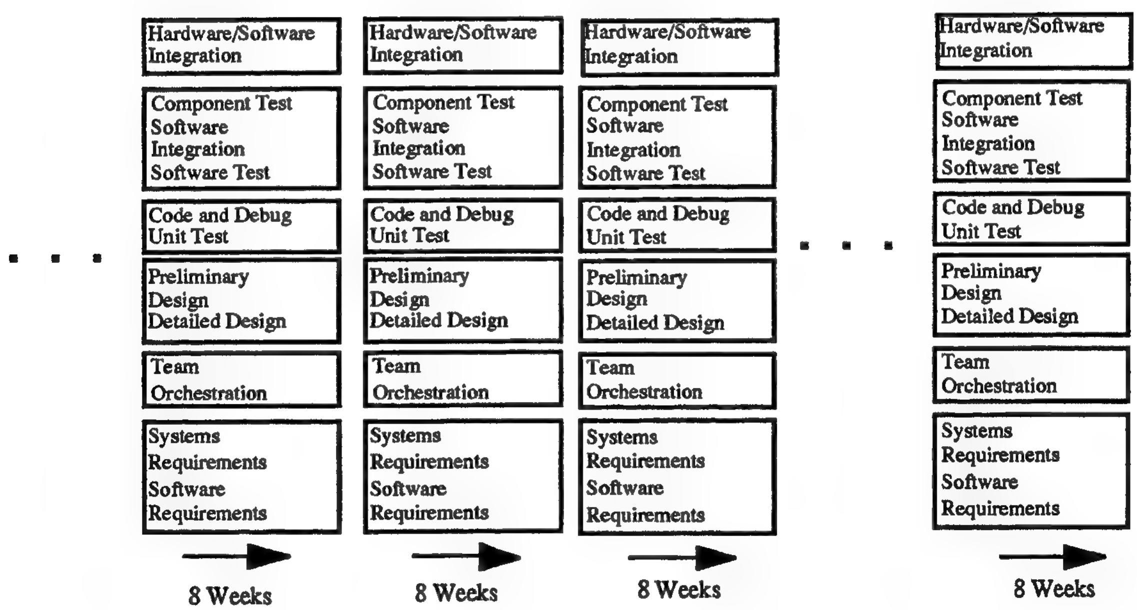

Figure 2. Parallel Development Activities

concentrates on maturing and further refining the functionality introduced in Phase 2. The goal of Phase 4 is a transition phase where the overall maintenance of the system and long-term viability are transitioned from JNIDS to the user.

The level of effort, as depicted by the shading in Figure 3 for each of the software development activities varies for each phase, with increasing concentration on system implementation as the effort moves through the phases to Phase 4.

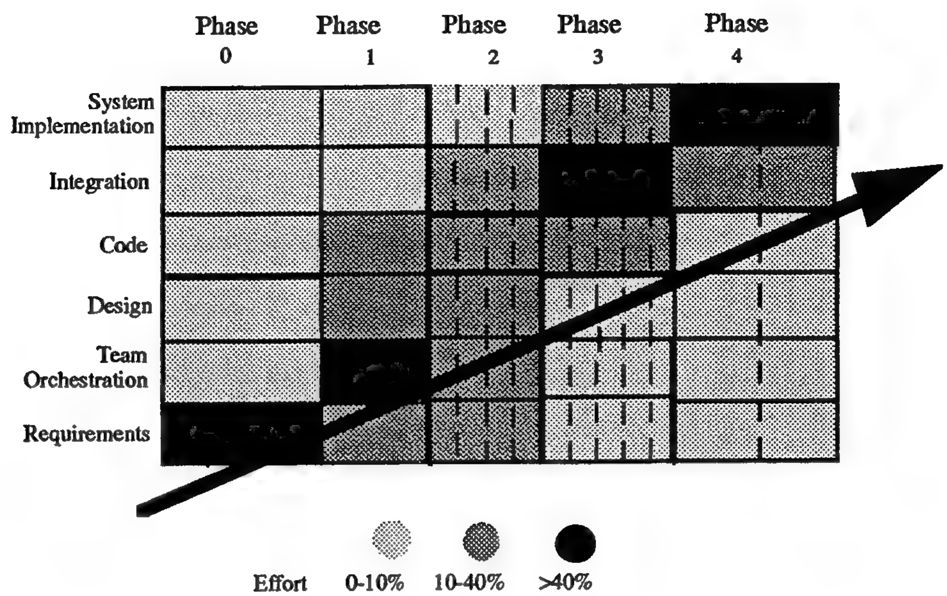

Figure 3. JNIDS Iterative Process

2.2.3 THE PRODUCT

JNIDS emphasizes the early delivery (within 6 months of project contract award) of an operational prototype in the user's environment. This delivery occurs at the end of Phase 1. Beginning with Phase 2, a step-wise strategy is established that determines intermediate requirements goals by delivering products incrementally at 8-week intervals. These intermediate goals are always aligned with and along a path toward the user's ultimate vision. The essence of product delivery centers around a process that repeats 2 significant meetings every 8 weeks: an iteration and an in-progress review (IPR). This repeating iteration cycle of meetings lasts 2 years and is depicted in Figure 4.

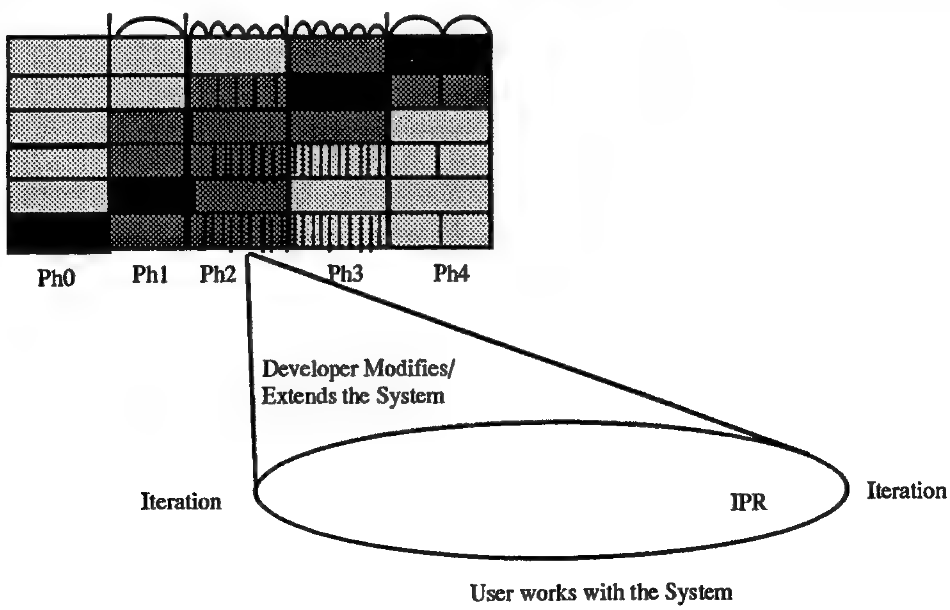

Figure 4. JNIDS Iteration Cycle

2.2.3.1 Iteration

An iteration is a technical and system-delivery meeting of representatives from the core team. The meeting typically lasts 3 to 5 days at the user's site. Holding this meeting at the user's site immerses the system developers in the user's work environment. Discussions during the course of the meeting focus on the functional and technical capabilities delivered. During the iteration, as the user learns, uses, and reviews the system, other needs and requirements are identified. By merging these new requirements with the previously identified requirements and factoring in cost and time schedules, the core team (JNIDS, users, and system developers) jointly agree upon a revised prioritized list of requirements by the end of the meeting. Every effort is made to include other participants and potential users through special demonstrations. This fosters organizational buy-in, gathers other potential support by expanding the user community, and publicizes the progress of the program beyond the core team.

2.2.3.2 In-Progress Review

An IPR is a management-oriented meeting that generally occurs at the development contractor's site. At the IPR, the core team discusses current progress, stressing budget/schedule management and program vision attainment. Future plans, developing problems or risks, and other management issues are also considered. An IPR typically occurs about 3 weeks prior to the next iteration/meeting. The IPR gives the user an important opportunity to meet and interact with the entire development contractor team, especially those who do not have an opportunity to attend an iteration/meeting.

With the system deliveries and management meetings as road markers on the path to the ultimate system delivery, the system developer continues, throughout the contractual period, to develop the system, and the user simultaneously works with and continually evaluates the system.

The notion of frequent meetings, iterative software development, and the opportunity for constant and continuous communication is not unique to JNIDS (Basili and Turner 1975). The Borland Quattro Pro for Windows (QPW) development group used a process that was "inherently iterative with a series of increasingly stable prototypes until the product was complete" (Gabriel 1994). They recognized that change management was crucial to successful design and implementation so they adopted "almost-daily meetings." While the JNIDS meetings were less frequent, the QPW and JNIDS reported similar benefits for holding these frequent meetings:

- Exchange of information to understand the other team member's contributions
- Appreciation for and sharing of information about your personal contribution to the "big picture"
- Collaborative, cooperative, and supportive environment achieved through shared dialogue
- Opportunity to reach mutual understanding and growth

2.2.4 THE SUCCESS

The JNIDS user-centric, iterative development process has proven successful on more than 11 projects, involving more than 25 development partners (from industry, academia, and national laboratories) and more than 20 user organizations worldwide. While some smaller project efforts were undertaken, the typical JNIDS project spanned 3 to 4 years, cost about $8M, and had a core team of approximately 25. In the words of one satisfied user, "from a pragmatic point of view, it is probably the best way for an . . . organization to obtain training, equipment, and expertise on newly emerging

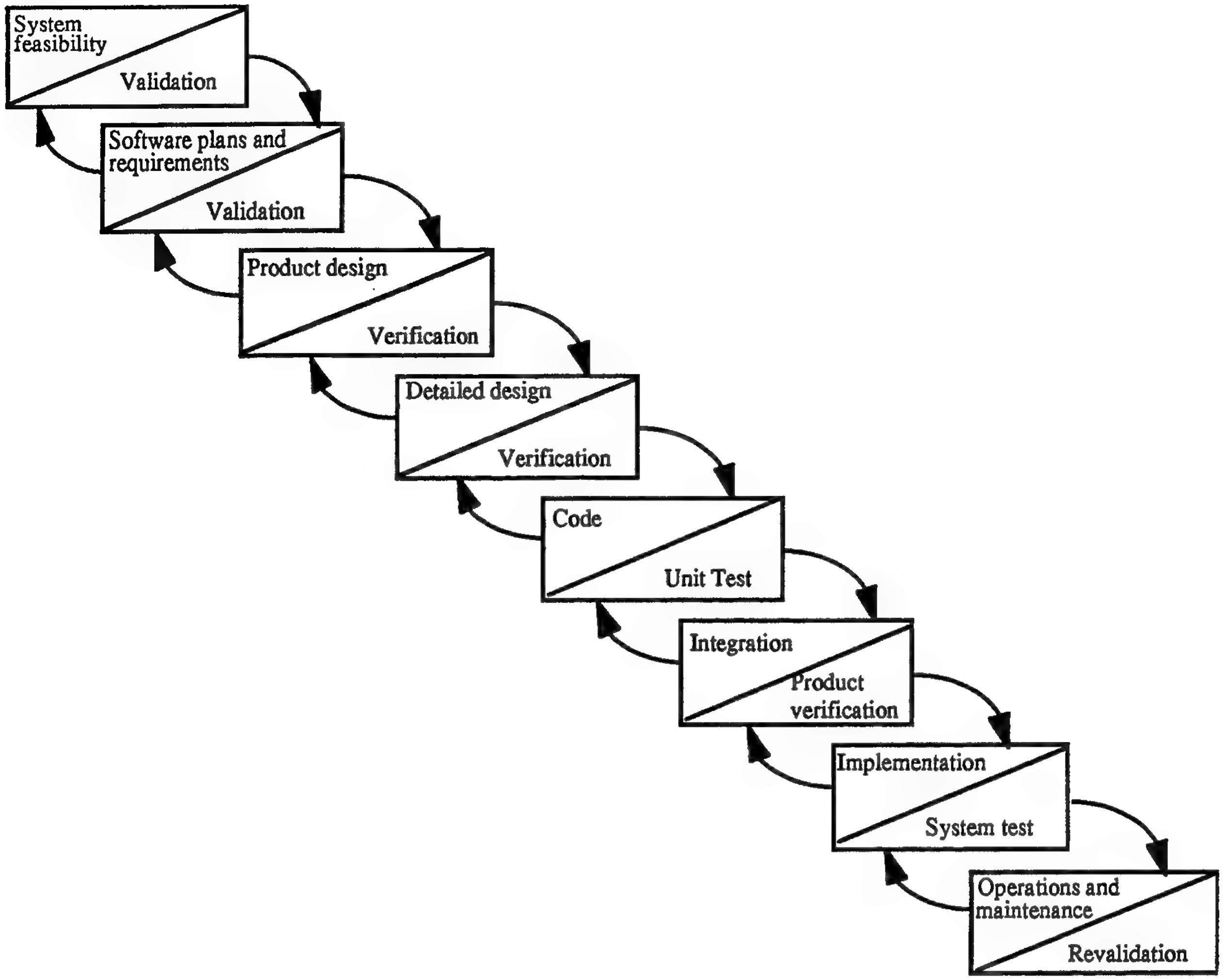

Figure 5. The Waterfall Process

technologies which in the long run will result in products and services which provide the very best . . . support possible" (JNIDS 1993).

2.3 SOFTWARE DEVELOPMENT TRENDS

Software development started as a waterfall. The traditional "waterfall" software development process (see Figure 5) has an initial requirements-acquisition phase during which the requirements are gathered, analyzed, negotiated, and rigidly defined. A user, although in many cases not the ultimate user, produces a paper-based requirements document. This requirements document initiates a chain reaction of documents that creates a paper trail through all of the remaining phases of software development. Except for the initial phase, the potential users of the system are essentially absent from the software development process. For large systems, there may be a multiyear gap between the time the user defines the requirements and the time the system is delivered to the users. The time delay and the user's changing environment renders the system obsolete upon delivery (DeBellis and Haapala 1994).

Most notably, the waterfall process fails in the following areas (Moad 1994):

- Failure to capture or keep up with real business requirements
- Little control over the waterfall process by business managers
- Lengthy development cycles

However, systems developers are no longer "slavishly following the (static) statement of requirements from users" (Moad 1994), but, more appropriately, working with users to develop a "sequence of transformations" (Wood and Sommerville 1988) that best solve the users' needs. The need to move in a more iterative direction occurred most recently when two trends converged.

Two contributing trends that prepared the way for UCD are:

- The failure of the waterfall methodology to satisfy customers by keeping up with changing business requirements
- The emergence of the client-server technology

2.3.1 Business Environment Trend

The business environment is the ultimate user. The last three decades have seen a movement in the types of systems that users are requesting. This relationship is depicted in Figure 6 as a move away from the financial-like, transaction-based systems of the early 70s to the more knowledge-based systems of the 90s—systems that process, synthesize, and facilitate the use of knowledge.

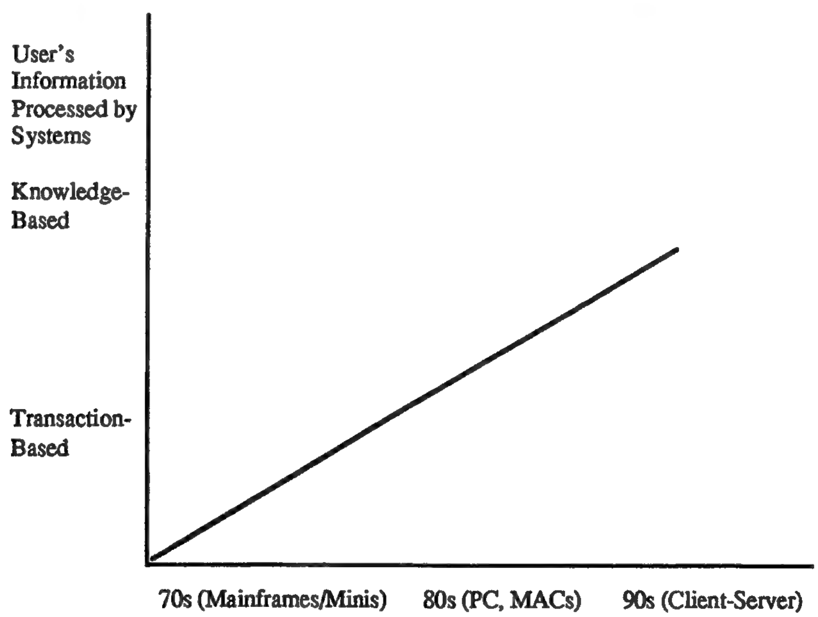

Figure 6. User's Information Processed by Systems

The "information society" user is clambering for instantaneous access to and use of knowledge. The "knowledge worker of the information based organizational" (Drucker 1988) needs systems designed for his specific requirements that will support collaboration, visualization, dissemination, integration, and cooperation. Not only do these systems have to fit today's business needs, but they must also "anticipate them" (Moad 1994).

2.3.2 CLIENT-SERVER TREND

The latest and largest trend in information technology is the "migration to client/server architecture" (Martin 1994). This trend is depicted in Figure 7. For example, in the 70s, a financial analyst would request a report though a systems shop and it would take a month or more to show up on his desk in reams of computer printouts. This process was slow, nonresponsive, and inflexible. The user's access to the information was through a system developer. In the 80s, with the advent of the PC and MAC, and their associated "user-friendly" software, this same report could be generated by the user in a matter of hours or minutes. However, access to the data was still an issue and resulted in numerous point-solutions with downloaded information, which quickly became outdated and out of sync with the rest of the organization. Integration was nearly impossible. Now, client-server technology brings the integration of the mainframe with access to large data sources and a streamlined front end with sophisticated and user-friendly software.

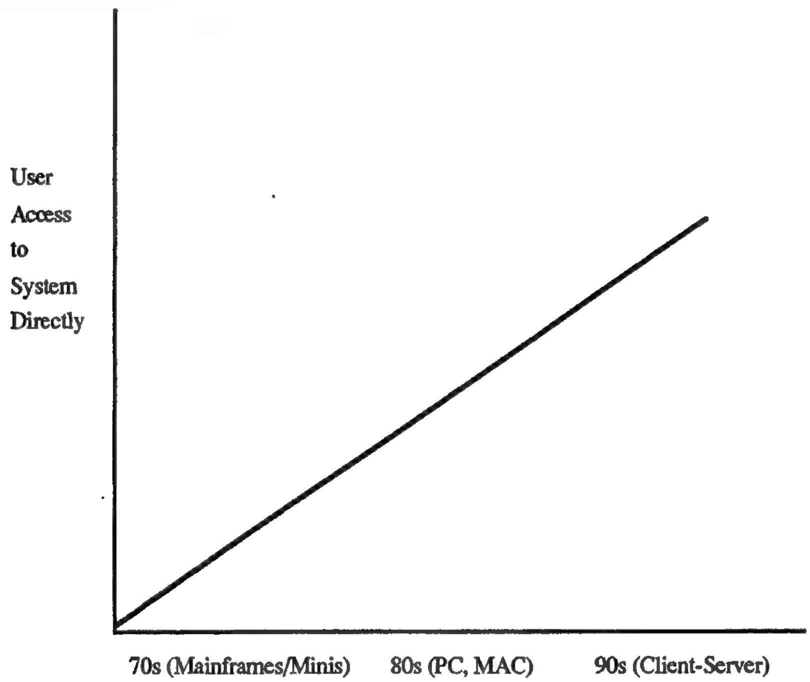

Figure 7. User Access to System Directly

The user needs to be involved in the system design of today's applications. Also, the "complexity of client-server applications is causing a shift in development methodologies from the waterfall method and toward rapid application development" (Adhikari 1994).

With the failure of the waterfall development process to satisfy the user and the advent of the powerful client-server technologies, UCD emerges. Only a "methodology with a high degree of user involvement would lead to better systems" for the users (Carmel, Whitaker, and George 1993). Because of client-server technology, we are being asked to become more business oriented and develop more new-technology expertise (Moad 1994). This fact, coupled with the "growing recognition that conventional, rationalistic systems development approaches are inadequate for dealing with the interpersonal aspects ..." (Clement 1994) has forced numerous solutions to alternative processes and methods to emerge over the last decade. Section 2.4 describes some of the user-focused methodologies that preceded UCD.

2.4 OTHER USER-FOCUSED METHODOLOGIES

There are two other significant user-focused development methodologies: Joint Application Development (JAD) (Wood and Silver 1989) and Participatory Development (PD) (Clement and Van den Besselaar 1993) that emerged during the same time that UCD emerged.

JAD and PD are well-known methodologies that demand user involvement and user participation. The goal of JAD is to "accelerate the design of information systems and promote comprehensive, high quality results" (Carmel, Whitaker, and George 1993). JAD is the fundamental methodological basis for Martin's Rapid Application Development (RAD) (Foss 1993; Carmel, Whitaker, and George 1993; Martin 1991) and "considers human factors and corporate culture as well as technology" (Adhikari 1994).

PD is the "Scandinavian approach" to systems development. It "advocates a much stronger form of user involvement than that of JAD" (Carmel, Whitaker, and George 1993). Additionally, PD strongly promotes a "mutual learning process between members of the group" and the "empowerment of workers so they can codetermine the development of the information system and their workplace" (Clement and Van den Besselaar 1993).

While UCD has much in common with JAD and PD, the most significant difference is that, in UCD, the user is central. In UCD, all development revolves around the user, the user's environment, and his ultimate needs. Figures 8 and 9 depict the distinctions among the four processes: UCD, JAD, PD, and the waterfall.

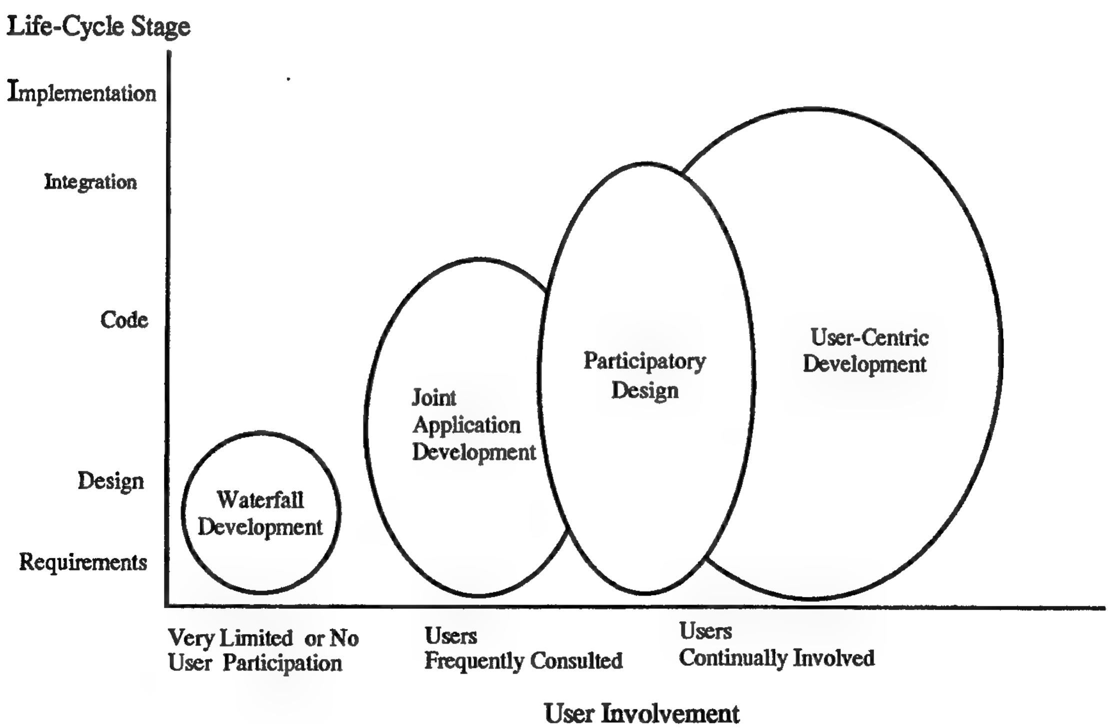

Figure 8. User Involvement in Four Processes

Some other user-focused methodologies include Codevelopment (Anderson and Crocca 1993), Contextual Design (Whiteside, Bennett, and Holtzblatt 1988), User-Centered Requirement Analysis (Martin 1988), User-Centered Design (Norman and Draper 1986), and IBM's Rapid Delivery.

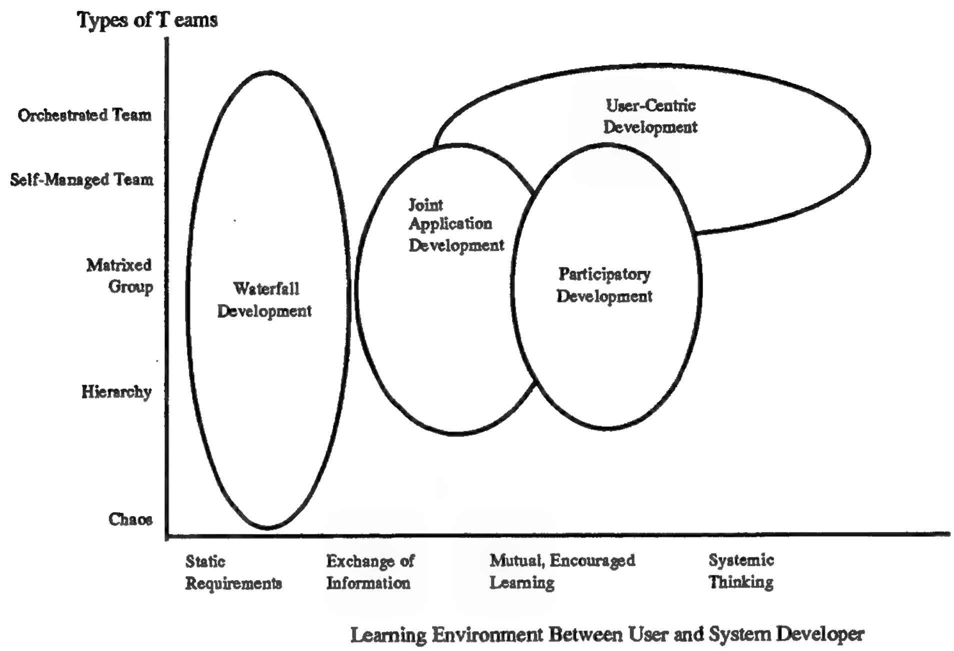

Figure 9. Mutual Collaboration Fostered by the Team Maturity Level

This page intentionally left blank.

3. USER-CENTRIC DEVELOPMENT PROJECTS

The plan for this case study called for a small number of software development projects that had successfully practiced UCD. The Consortium hoped to have some variety in the projects to provide evidence of how widely applicable UCD might be. The Consortium found four organizations that had practiced and institutionalized UCD across a number of projects for 5 to more than 10 years. These organizations included the research arm of a commercial developer, a federal agency, and two organizations within the Department of Defense (DoD). One thing all these organizations had in common is that individuals took personal, career risks to pioneer UCD in their organizations. The practice of UCD was not officially sanctioned or supported by the organizations and was sometimes opposed by some of the organizations.

The Consortium also found several DoD contractors that had institutionalized UCD to some extent. The developers on projects practicing UCD were separated from the contractors'other projects.

The projects the Consortium interviewed for the case study represented a range of applications and a number of different types of organizations in the roles of user, customer, and developer. Table 1 illustrates the combination of roles that organizations played in the projects in this case study. Many of the projects developed analytical support workstations. These were typically knowledge- and data-intensive and required a real-time response. Several of the projects developed information systems to be used by occasional, sometimes one-time, untrained users. A big part of the challenge of building such systems was accommodating people who had never used a personal computer and were unfamiliar with typical user-interface conventions (e.g., using a mouse to point and click). A number of the projects built parts of hospital information systems. Some of the applications involved collecting, verifying, and reporting lab results and scheduling appointments. A series of projects involved building systems to support planning the transportation of supplies during war.

Table 1. User-Centric Development Roles Played on Projects

User	Customer	Developer
DoD	DoD	Commercial firm
Federal agency	Federal agency	Federal agency
Individuals	Commercial firm	Commercial firm
Individuals	Individuals	Commercial firm
Commercial firm	Commercial firm	Commercial firm
DoD	Federal research and development	Commercial firm

This page intentionally left blank.

4. CONDUCT OF THE CASE STUDY

The case study is modeled on that described in (Yin 1989). In Yin's model, researchers pose questions they want to answer. Based on observation and the scientific literature, the researchers develop hypotheses, called "propositions," which propose answers to the research questions. They test the propositions by conducting interviews, reviewing documents, etc. From the results of the tests, the researchers draw conclusions about whether the evidence supports, denies, or neither supports nor denies the hypotheses. These conclusions provide some answers to the research questions they originally posed.

The Consortium's interest in understanding UCD, how better to transfer it, and what other technologies might leverage it drove the development of the research questions listed in Tables 2, 3, and 4. The tables organize the questions into those that compare UCD with other software development processes, factors that affect adoption of UCD, and factors that support UCD.

Table 2. Case Study Questions: Comparison of User-Centric Development With Other Processes

1.	How does the UCD process differ from other software development processes?
2.	What are the distinguishing characteristics of the UCD process?
3.	What are the advantages and disadvantages of UCD over other software development processes?
4.	How do the products delivered by the UCD process differ from the products delivered by other software development processes?
5.	What are the distinguishing characteristics of the teams that practice the UCD process?
6.	How are the products of the UCD process delivered to users?

Table 3. Case Study Questions Adoption of User-Centric Development

1.	What conditions would inhibit adoption of UCD?
2.	What will facilitate widespread use of UCD?
3.	What technological advances will encourage UCD?

Table 4. Case Study Question: Support for User-Centric Development

1.	What technological advances will leverage UCD?

The Consortium developed propositions that propose answers to the research questions. Section 5 presents and discusses the propositions. For each of the propositions, the Consortium developed one or more interview questions (Appendix A presents the interview quesitons) about UCD projects that would confirm or refute the proposition. For example, to test the first proposition in Section 5:

> User-centric software development is a process. It can be successfully practiced with different methods and tools.

The Consortium posed the following questions about UCD projects in its interviews:

- What software development methods did you use (e.g., structured design, structured analysis, object-oriented design)?
- What tools did you use with each method?
- How did the tools support your development process?

Sixty-nine such questions drove interviews the Consortium conducted with participants in UCD projects. Seventeen individuals answered the interview questions. Four of the individuals provided written answers to the written interview questions. The other 13 individuals provided oral answers in interviews conducted by the Consortium. One interview was conducted solely by one author. The rest of the interviews were conducted by both authors. One interview was of three individuals, three interviews were of two individuals each, and four interviews were of one individual each. The Consortium used notes from the interviews and the written answers from the four individuals to evaluate the propositions.

Section 6 discusses conclusions the Consortium drew from the interviews and written responses to the interview questions.

5. USER-CENTRIC DEVELOPMENT PROPOSITIONS

Section 3 discusses the research questions about UCD that the Consortium wanted to answer. The questions related to how UCD compared with other software development processes, factors that would encourage or inhibit organizations to adopt it, and technologies that would leverage it. The approach the Consortium took to answering these questions was, first, to develop hypothesized answers to the questions in the form of a set of propositions. Then, the Consortium interviewed a number of individuals who participated in UCD projects as users, customers, or developers. Comparing notes from the interviews to the propositions provided evidence the Consortium used to answer the research questions.

Table 5 lists propositions that characterize the management of projects practicing UCD and the software process that the projects follow. The propositions hypothesize that UCD projects are less risky, need less management, and can use a variety of software development methods and tools.

Table 5. Management and Process Propositions

1.	UCD is a process. It can be successfully practiced with many different methods and tools.
2.	UCD gives management confidence that a project is progressing (as opposed to simply moving).
3.	Several low-cost iterations reduce the need for executive involvement.
4.	UCD projects encourage risk identification and facilitate resolution.

The propositions in Table 6 hypothesize that UCD projects deliver products that are more useful, usable, and better reflect the users' current needs.

Table 6. Product: Customer Focus Propositions

1.	UCD leads to the delivery of a more useful and usable product.
2.	UCD projects tend to address current requirements.

Table 7 lists propositions that characterize the nature of UCD work products (e.g., requirements, design). The propositions hypothesize that user-centric projects detect problems earlier; deliver more current technology; produce more complex designs; and support rapid change and construction of software work products, facilitating the practice of UCD.

Table 7. Product: Developer Focus Propositions

1. UCD leads to earlier detection of problems in requirements and design.
2. UCD can lead to more complex designs.
3. UCD projects tend to deliver current technology.
4. Methods that support rapid change and construction of software work products will facilitate the practice of UCD.

Table 8 lists propositions that hypothesize that teams are critical to the practice of UCD and that practicing UCD leads to the formation of teams.

Table 8. Team Propositions

1. UCD leads to the formation of a team.
2. UCD is effective because a team forms.
3. Many short-term deliverables lead to team formation.
4. Participants in UCD tend to be more motivated.
5. Teams use peer pressure to keep members "in line."
6. There is a low personnel turnover rate in UCD projects.
7. To be effective, UCD requires a balanced team.
8. UCD fosters personal and technical development in team members.
9. The rewards for members of a UCD project include being part of a successful team and delivering a useful product.

The propositions in Table 9 hypothesize that UCD is an effective means of transferring technology both into and out of the project.

Table 9. Technology Transfer Propositions

1. UCD leads to user buy-in of the product.
2. UCD projects, by nature, tend to attract resources (magnet effect).
3. Groups participating in UCD tend to "market" to their peers.

6. RESULTS OF THE CASE STUDY

This section presents the results of comparing the UCD propositions discussed in Section 5 to notes taken of interviews with individuals from organizations that practice UCD. Annotations on the tables of propositions in Section 5 summarize the results. A plus sign (+) in the column preceding a proposition indicates that the interviews tended to support the proposition. A minus sign (-) in the column preceding a proposition indicates that the interviews did not tend to support the proposition; that is, the individuals interviewed made comments contradicting the proposition, made no comments relevant to the proposition, or the individuals making comments relevant to the proposition tended to contradict one another.

6.1 SUPPORT FOR MANAGEMENT AND PROCESS PROPOSITIONS

Table 10 lists the propositions from Table 5. The interviews tended to support all of these propositions. In support of the first proposition, the organizations reported using a wide range of software development methods and tools. Methods used included none, ad hoc and in-house-developed methods, Structured Analysis/Structured Design, a standard set of abstractions that separated policy concerns (e.g., all orders over $5,000 must be approved by a Vice President) from mechanism concerns (a manager approves an order by sending an e-mail message to a particular address), object-oriented design, and RAD. Tools used included none, in-house tools (including code generators), and commercial Structured Analysis/Structured Design tools.

Table 10. Support for Management and Process Propositions

+	1. UCD is a process. It can be successfully practiced with many different methods and tools.
+	2. UCD gives management confidence that a project is progressing (as opposed to simply moving).
+	3. Several low-cost iterations reduce the need for executive involvement.
+	4. UCD projects encourage risk identification and facilitate resolution.

While the second and third propositions in Table 10 were supported by the interviews, the reader should note that the UCD projects in the interviewed organizations did not appear to be treated any differently than other projects in those organizations. One interviewee noted that "[one customer organization represented in these interviews] picked smaller contractors, or parts of larger contractors that acted like their own company."

Most of those interviewed stated that they could not have written the requirements for the systems built using UCD before actually building versions of the systems. This statement supports the fourth proposition. Risks that UCD appears to address particularly well are eliciting, understanding, and confirming users' needs with them. One interview provided a particularly clear example of this:

> [Company] wrote the proposal. The users loved the proposal. [Company] built the product. The users hated the product. So [Company] went off and rebuilt the product. The users love the rebuilt product.

The reader should note that not all risks a system might encounter are equally likely to be identified. Those risks that are most likely to be identified and addressed are those that surface during daily use of a system for routine tasks.

6.2 SUPPORT FOR PRODUCT: CUSTOMER FOCUS PROPOSITIONS

Table 11 lists the propositions from Table 6 The interviews tended to support both of these propositions. The report by most interviewees that they could not have written the requirements before building the system supports both of the propositions. Many interviews reported examples of requirements that the developers discovered by having the users use the product. One user gave particularly clear support for the first proposition by contrasting two products with which he was familiar: [Product 1] was developed following a traditional waterfall process; [Product 2] was developed by a team practicing UCD.

> [Product 1] was driven by requirements only; it's hard to use. [Product 2] users don't have to open lots of windows to enter one name.

He went on to note:

> The convenience of data entry [in Product 2] is a reflection of the users selected: experienced analysts.

This latter comment brings up a corresponding risk that several interviewees mentioned. The system that the UCD team produces reflects the concerns, abilities, and interests of the users participating in development. Misguided users can lead the project astray.

Table 11. Support for Product: Customer Focus Propositions

+	1. UCD leads to the delivery of a more useful and usable product.
+	2. UCD projects tend to address current requirements.

In support of the second proposition that UCD addresses current requirements, one developer stated:

> The target [of the product] was constantly changing. There is no way you could have written requirements ahead of time [would have had to be very flexible to do so].

6.3 SUPPORT FOR PRODUCT: DEVELOPER FOCUS PROPOSITIONS

Table 12 lists the propositions from Table 7. The interviews tended to support all of these propositions. The interview comments, discussed in Section 6.2, that the teams could not have written the requirements for the systems before building them, support the first proposition. Several interviews gave examples of users finding delivered products to be unusable or inadequate and of

developers responding by producing new versions that better addressed users' needs. One developer, describing a particular project, reported:

> People who were too junior were trying to use a tool that was too sophisticated. About one and a half people were able to use it. Near the end of the project, we went to a self-propagating knowledge base because the junior people could not provide the information that the tool needed.

Table 12. Support for Product: Developer Focus Propositions

+	1. UCD leads to earlier detection of problems in requirements and design.
+	2. UCD can lead to more complex designs.
+	3. UCD projects tend to deliver current technology.
+	4. Methods that support rapid change and construction of software work products will facilitate the practice of UCD.

The evidence in support of the second proposition was that those who did not find UCD leading to more complex designs described techniques they used to keep constant changes to the design from making it increasingly complex. Several interviews reported using very senior people, who had years of experience in the application area, to develop the designs. One interview reported that they refined the design and code over time. Another stated:

> We took a modular approach, kept things open. We over-designed the first few iterations because we didn't know which way we were going to go.

Reports by a number of interviews that technology was added during the course of the project, sometimes quite late, support the third proposition. The inclusion of the self-propagating knowledge base is one example of a technology being added late in a project.

The nature of UCD, that is, developers need to produce and change versions of the product, supports the fourth proposition. The discussion of how developers kept their designs from becoming complex due to changing requirements also supports the fourth proposition.

6.4 SUPPORT FOR TEAM PROPOSITIONS

Table 13 lists the propositions from Table 8. The interviews tended to support seven of the nine propositions. The reports by all the interviews that teams were important to the practice of UCD, and that teams had formed support the first proposition. From the evidence available, it is hard to know to what extent UCD led to team formation and to what extent it was in the nature of the participating organizations that teams would form. Certainly, for many of these organizations, teams appeared to be a natural way to organize.

> [One customer organization represented in these interviews] picked smaller contractors or parts of larger contractors that acted like their own company.

Several interviews noted difficulties forming teams of teams. One developer on a multiple contractor project noted:

> [Project] was not a team... There was a lack of a strong leader.... In many cases, [Company] was not treated as an equal member of the team.

Another developer stated:

> On the small teams, people do take responsibility for the project and form a team. I don't understand the mechanisms for coordinating teams of teams. I haven't seen a lot of teams of teams' success.

Table 13. Support for Team Propositions

+	1. UCD leads to the formation of a team.
+	2. UCD is effective because a team forms.
+	3. Many short-term deliverables lead to team formation.
+	4. Participants in UCD tend to be more motivated.
-	5. Teams use peer pressure to keep members "in line."
-	6. There is a low personnel turnover rate in UCD projects.
+	7. To be effective, UCD requires a balanced team.
+	8. UCD fosters personal and technical development in team members.
+	9. The rewards for members of a UCD project include being part of a successful team and delivering a useful product.

The fact that all of the interviews reported having teams supports the second proposition. The two instances where interviews reported problems with teams of teams and project problems support the second proposition.

Comments made in a number of interviews support the third proposition. One developer stated:

> Iterations travel is a good team-building thing. Delivering something that the user likes gets you out of the storming phase.

This statement supports the proposition for building a team of developers. The following statement by a user supports the proposition for a team of teams:

> [Developers and users] live and breathe together. Developers did their work in the middle of the analysts' area. Developers built up credibility in the first 6 months; developers delivered what was promised.

Some of the evidence that supports the fourth proposition is not communicated by written transcripts of the interviews. The motivation of the interviewees was obvious in the enthusiasm they showed for the products they were building and using and for the UCD process they were following. Their motivation was also evident in their descriptions of the most rewarding part of participating in a UCD project. Some typical comments were:

> Interesting work kept people on projects.... People were motivated by what they were doing.

[Most rewarding part of participating on this project is that] end users were able to do work because of work that I did.

There was little evidence to support the fifth proposition. The only instance of the use of peer pressure that the Consortium recorded was in a multiple-contractor environment. One contractor did the work that the other contractor could not do. A developer from the first contractor spent a week and a half at the other contractor's site to do the work.

There was mixed evidence in support of the sixth proposition. Several interviewees reported that they kept teams of 5 to 8 developers together for 2 to 4 years. Some reported starting projects with very senior developers to get things started and then bringing in junior people to replace the senior people who were assigned to new projects. Some of the user organizations (particularly military ones) turn over personnel more frequently. User turnover hurt at least one development project, when experienced senior users were replaced by junior ones.

Several of the interviews with development organizations supported the seventh proposition. One stated:

All team members must be very good tool developers. It's hard to carry a weak person on small teams... Need software cycle generalists. On small teams need very flexible people. It's hard to support people with specialties. They need to do everything that everyone else does.

User and customer interviews did not express such concerns.

While the interviews provided evidence to support the eighth proposition, it is not clear how much UCD projects differ from other projects in fostering personal and technical development in team members. One developer noted:

I've seen a lot of growth in new kids; expect it on any project.

Another developer commented:

[There are] lots of examples of people blossoming on a team. People are given lots of responsibility, but there is also a safety net... [We] use participation in existing teams to teach mores and values of teams: teach by example.

Many of the interviews supported the ninth proposition. Developers reported that the most rewarding part of participating in a user-centric project were:

...to build things that people will use

...looking at beautiful things on a screen that people might use

...end users able to do work because of work that I did

6.5 SUPPORT FOR TECHNOLOGY TRANSFER PROPOSITIONS

Table 14 lists the propositions from Table 9. The interviews tended to support all of these propositions. The following comment, quoted in Section 6.1, both supports the first proposition and gives some insight into why the UCD process leads to user buy-in:

[Company] wrote the proposal. The users loved the proposal, [Company] built the product. The users hated the product. So [Company] went off and rebuilt the product. The users love the rebuilt product.

For a particular project, another developer noted:

> The customer didn't believe the system would work.

The company was able to convince the customer by producing a working system. The same developer noted:

> We do better interacting with line of business manager at a customer than with the customer's manager of MIS....

One developer cautioned:

> Getting users involved can be an issue. Users are overburdened. Some users are too naive to understand what they should be doing.

Table 14. Support for Technology Transfer Propositions

+	1. UCD leads to user buy-in of the product.
+	2. UCD projects, by nature, tend to attract resources (magnet effect).
+	3. Groups participating in UCD tend to "market" to their peers.

The interviews provided some evidence to support the second and third propositions. A commercial developer reported that their customer brought in a system the developer built for the customer's internal use. The customer now sees the system as being a new business for them. Several contractors reported technologies or systems they built on contract were turning into products for other customers, sometimes commercial ones. One government user reported the following about a product developed by a UCD project:

> People who see it, want it. [One government organization] wants the technology. [Another government organization] has asked for adaptations to the product.

7. CONCLUSIONS AND FUTURE WORK

7.1 CONCLUSIONS

Section 6 discusses the evidence the Consortium gathered to address the research questions posed in Tables 2, 3, and 4 of Section 4. This section summarizes the evidence and draws some conclusions from it. The first four questions in Table 2 concern how UCD compares to other software processes. The evidence gathered indicates that UCD gives management confidence that the project is progressing, reduces the need for executive involvement, and encourages risk identification and resolution. The evidence also indicates that products of UCD projects are more usable, tend to address more current requirements and technology, and tend to detect problems in requirements and design earlier in the development cycle.

The major disadvantage of UCD to which the evidence points is that UCD projects tend to develop more complex designs if the projects do not make efforts to address the issue. The Consortium believes that this complexity is due to the frequent changes to the software, throughout its development, that developers make in response to developing an understanding of user needs. UCD could leverage work on design for change, maintainability, and reuse that has been going on for several decades (e.g., [Parnas 1979]). Several projects reported that they were attempting to anticipate how the software was likely to change over the course of the project and building the software in ways that would make the anticipated changes easier to make. An organized effort on a project to work with the users and customers to understand how the requirements for the software are likely to change and then to design the software to make those changes easily could result in software development that was more responsive to user needs. Changes could be made more quickly, the design would be less likely to become more complex as the developers respond to changing understanding of user needs, and the opportunities for obtaining leverage from reuse would increase.

7.2 FUTURE WORK

7.2.1 THE CURRENT ENVIRONMENT

The current business and technological environment is a sea of chaos with changing, evolving requirements and lightning speed introduction of new technologies. While UCD is designed to handle this influx of new, evolving requirements and take into account emerging technologies, there, however, is a need to manage and measure this change. Specifically, there is an overall need to understand how these requirements and technology affect the ultimate system implementation. To ensure a smooth progression of the introduction of requirements and technology into a UCD effort, there is a need for techniques, methods, and tools to:

- Assess the impact of the insertion of the new requirements or the technology's contribution in satisfying the long-term vision.

- Insert the new requirements or technology and maintain the integrity of the design and implementation of the system.
- Recognize, at any state in time, the requirements that are satisfied by the current version of the software. While this is a particular need to determine the state of work in progress, this is a critical requirement for determining suitability to transfer this software to other users or other domains.
- Define, divide, and measure the individual iteration's (system build) functionality so that work in progress at each incremental delivery meets or exceeds expectations.
- Measure relevant UCD success indicators or factors simultaneously with the iterative cycle but completely nonintrusively to the programming and development cycle.
- Decompose and measure individual functionality to fit into the iterative cycle framework.

7.2.2 Looking to the Future

UCD requires a future visioning mindset. Because the requirements and needs are typically unknown or difficult to articulate by the user, there needs to be a vision, or a long-term goal, for which the team strives. Because the vision is the unifying glue that brings the team together, there is a need for techniques, methods, and tools to:

- Define, articulate, and develop the team (and organizational) buy-in to the vision.
- Perform the UCD effort in anticipation of the future. To think beyond today by acting with a long-term focus in mind.
- Read trends, both business and technology. Ability to develop scenarios and predict outcomes based on these trends.

7.2.3 Supporting Multiple Users

The economic realities of limited budgets and the end of building one-of-a-kind systems require that software serves more than one user. Because of the nature of UCD's ability to respond quickly to a changing environment, and given that the changing environment requires adding users, there is a need for techniques, methods, and tools to:

- Assess the cost, performance, effect on schedule, and benefits of taking a partial solution within the iterative development and transferring that solution to another user or another domain, now or sometime in the future
- Understand the implications of supporting multiple users and multiple domains at the same time as developing the system for the primary users; that is, to simultaneously conduct an UCD effort while expanding the user base to ensure long-term survivability and sustainability of the project
- Market the project by finding efforts that can benefit from the technology built and going to be built
- Find the group of users that will take a specific application and collaborate on a general application with extensions to match each of the user's specific needs

7.2.4 TEAMS

UCD is performed by orchestrated teams. UCD is dependent on, but fosters, the successful coalescing of a team and the emergence of a leader. UCD is a perfect environment for a team and leader to emerge; but it takes time, work, and attention. There is a need for techniques, methods, and tools to:

- Grow and nurture UCD teams
- Identify and grow UCD leaders
- Facilitate communication among users and developers
- Facilitate long-distance communication among users and developers
- Quickly form high-performing, self-directed teams; produce significant and lasting results; dismantle teams; and reform teams
- Work in a cooperative, collaborative fashion across domains
- Work together within a significant degree of uncertainty

7.2.5 VERIFICATION AND VALIDATION

Current verification and validation (V&V) techniques are designed for the waterfall process under conditions of certainty and static requirements. With the current techniques, in a constantly evolving requirements scenario, V&V can be nothing more than a snapshot of the state of the system. There is a need for techniques, methods, and tools to:

- Conduct V&V in an evolving environment
- Perform V&V on a "potential" system, one yet to be built

7.2.6 SPEEDING OF THE CYCLE

UCD typically iterates and builds interim systems in under 2 months. This is still not fast enough. There is a need to increase the speed of UCD development through techniques, methods, and tools to:

- Satisfy user requirements in real time (not to be confused with real-time systems) as the user's requirements are changing hourly and daily

This page intentionally left blank.

APPENDIX: QUESTIONNAIRE

USER-CENTRIC SOFTWARE DEVELOPMENT QUESTIONNAIRE

JUNE 9, 1994

The following questions are intended to be answered about one particular software development project that is practicing (or has practiced) UCD. The identifying characteristics of UCD are:

- The team that develops the software includes some of those who will use the software.
- The team uses iterative enhancement to develop the software: an early limited version is delivered early; feedback from users is used to drive the development of a series of versions.

This survey is part of an ARPA-sponsored study of UCD by the Software Productivity Consortium. Responses to this questionnaire and questions about this study should be directed to:

Software Productivity Consortium
2214 Rock Hill Road
Herndon, VA 22070
(703) 742-7217/7152
FAX (703) 742-7200
e-mail: ask-spc @software.org

QUESTIONS ABOUT YOU AND YOUR PROJECT

1. How can we contact you?
   - Name and title
   - Organization
   - Address
   - Telephone #
   - FAX #
   - E-mail address

2. What was your role on this project (e.g., developer, developer and user, user, project manager, executive management, other)?

3. What is the name of your project?

4. What is your best estimate of the size of the software product on final delivery in lines of source code?

5. For how many months has your project been underway?

6. Does (did) your project practice UCD?

   If yes, answer the following:

   a. How many iterations have been delivered to the customer to date?

   b. How long was the typical time between delivery of iterations?

QUESTIONS ABOUT THE SOFTWARE PROCESS YOU FOLLOWED

7. What software development methods did you use (e.g., structured design, structured analysis, object-oriented design)?

8. What tools did you use with each method?

9. Have you used these methods and tools before?

10. What criteria were used in deciding to use these methods and tools? Were the methods and tools more a function of the product to be delivered or a function of the UCD process?

11. Would you use these tools/methods again?

12. How did they support your development process?

13. Are you aware of any tools/methods that are available now that would be more appropriate than the methods and tools that you used?

14. If you had to do it again, would you? Would you with the methods and tools that you used?

15. How was your development team organized? Draw the formal management structure. Draw the information management structure.

16. How often did you meet with your formal management structure?

17. What was the nature of the meetings?

18. How did the nature of these meetings change as the project progressed?

19. What were the monthly (or whatever) reports or status checks on the project? From within the project? From without the project?

20. Were the same management oversights performed on this project as opposed to other or similar projects within the corporation/organization?

21. Was the management oversight reasonable? Unreasonable?

22. Compare this management oversight with previous projects you have been involved in.

23. If you were to do this over, what would you do differently in regard to the management oversight of the project?

24. Did your project stay on-time and on-budget?

    If so, then how did you stay on-time and on-budget?

    If not, when did you discover that the project was not going to stay on-time and on-budget?

25. Did your project deliver the functionality that it promised?

    If so, then how did you do it?

    If not, when did you discover that you weren't going to deliver what you promised?

26. Compared to other projects that you have participated on, was the risk handling the same or different? How was it the same or different?

27. If you were to do this over, how would you handle risk identification and facilitation?

QUESTIONS ABOUT THE PRODUCT: CUSTOMER FOCUS

28. Can you identify a need that the product satisfies that you hadn't anticipated prior to the project?

29. Do you think that you could have written the requirements for this system before building it?

30. Are there requirements that you identified for this system that the system does/will not satisfy? If so, what are they and why were they not satisfied?

31. Have the product and the user's business process evolved together?

32. What is the customer's evaluation of the product?

33. How is the customer using the product?

34. What can the customer do now, because of the product, that he couldn't do before?

35. Describe the project from the customer's point of view.

36. Describe the product from the customer's point of view.

37. Describe the work products that supported the customer.

QUESTIONS ABOUT THE PRODUCT: DEVELOPER FOCUS

38. Did the requirements for the product and its design evolve together?

39. If you'd known at beginning of development what product you wanted to build, could you have produced a simpler design for the product than you did?

40. Did you find commercial off-the-shelf (COTS) tools useful in adapting requirements, design, implementation?

41. Describe the project from the developer's point of view.

42. Describe the product from the developer's point of view.

43. Describe the work products that supported the developer.

44. For technology that was added over the course of the project (e.g., new hardware such as display terminals or sensors, new software subsystems such as a database management system), who originally identified it?

45. Was all the technology used in the product available at the start of the project?

46. What technology was delivered with the project?

47. When?

48. Did the technology change over the course of the project? If so, how? What factors were used to warrant the change?

49. What has been your experience with adapting the product to changing requirements? Has there been any change in the effort required to adapt the product? How would you characterize that change? What do you think is the cause of the change?

50. Based on the original and current designs of the product, are there significant differences between the two? What is the cause of those differences?

51. If there were requirements changes, how was the design affected? Characterize what you think made the effects on design major or minor.

QUESTIONS ABOUT THE TEAM THAT BUILT THE PRODUCT

52. What is your definition of a team?

53. Has a team formed?

54. How do members of the team interact?

55. Describe the team: Who are the members? From what organization? What were the roles of each of the members?

56. Were the team members ever on a UCD project?

57. Were any of the team members ever a member of the same team before? Explain.

58. Was the growth of the team smooth?

59. Were there problems with particular individuals or organizations?

60. How were these problems resolved?

61. Detail the growth of the team. Any significant turning points in the team formation (i.e., loss or addition of team member, corporate restructuring, etc.)?

62. Were there any extraordinary team building exercises (i.e., Outward Bound to facilitate quick forming of the team)?

63. Was the project able to deliver useful functionality in a timely manner to the user?

64. What was the schedule for delivering products to the user?

65. How hard was it to meet that schedule?

66. Do the team members seem motivated?

67. Do they want to talk about their work on the project?

68. Describe the makeup of the team. How long has each member been part of the team? part of the company?

69. How has an individual's role in the team changed? What is he doing that he hasn't done previously?

70. Have you seen any remarkable change in team members or in the team as a whole?

71. Have you participated in teams like this before? Under what conditions?

72. What is the most rewarding part of participating on this project?

73. What is the least rewarding part of participating on this project?

QUESTIONS ABOUT TECHNOLOGY TRANSFER OF YOUR PRODUCT

74. How has the user base for the project grown or changed?

75. Have there been requests for adaptations of the project's products from groups that were not previously users of the products?

This page intentionally left blank.

LIST OF ABBREVIATIONS AND ACRONYMS

ARPA	Advanced Research Projects Agency
COTS	commercial off-the-shelf
DoD	Department of Defense
IPR	in-progress review
JAD	Joint Application Development
JNIDS	Joint National Intelligence Development Staff
PD	Participatory Development
QPW	Quattro Pro for Windows
RAD	Rapid Application Development
UCD	user-centric development
V&V	verification and validation

This page intentionally left blank.

REFERENCES

Adhikari, R.
1994
Planning, Testing, Teamwork a Recipe for Quality Applications. *Software Magazine*.

Anderson, W., and W. Crocca
1993
Engineering Practice and Codevelopment of Product Prototypes. *CACM*.

Basili, V., and A. Turner
1975
Iterative Enhancement: A Practical Technique for Software Development. *IEEE Transactions on Software Engineering*, Vol. SE-1, No. 4.

Brooks, F.
1972
The Mythical Man-Month, Essays on Software Engineering. Addison-Wesley Publishing Co.

Carmel, E., R. Whitaker, and J. George
1993
PD and Joint Application Design, A Transatlantic Comparison. *CACM*. June 1993.

Clement, A., and Van den Besselaar
1993
A Retrospective Look at PD Projects. *CACM*. June 1993.

Clement, A.
1994
Computing at Work: Empowering Action By 'Low-Level' User. *CACM*. January 1994.

Constantine, L., and L. Lockwood
1993
Orchestrating Project Organization and Management. *CACM*. October 1993.

Davis, A., E. Bersoff, and E. Comer
1988
A Strategy for Comparing Alternative Software Development Life Cycle Models. *IEEE Transactions on Software Engineering*, 14:10.

DeBellis, M., and C. Haapala
1994
User-Centric Software Engineering, to be published in *IEEE Expert*. 4th Qtr, 1994.

Drucker, P.
1966
The Effective Executive. New York, New York: Harper & Row.

1988
The Coming of the New Organization. *Harvard Business Review*. Jan-Feb., 1988.

Foss, W. 1993	Fast, Faster, Fastest Development. *Computerworld.* May 31, 1993.
Funk, J. 1992	The Teamwork Advantage An Inside Look at Japanese Productivity and Technology Development. *Productivity Press, CT.*
Gabriel, R. P. 1994	Productivity: Is There a Silver Bullet? *JOOP.* March-April 1994.
Haapala, C. 1992	"Rapid Application Through Iterative Development." in *Software Improvement Conference.* November 1992, Washington, D.C.
1993	Establishing Lasting Partnerships. MANAGE, The National Management Association Magazine. April 1993.
Haapala, C., C. Hess, and R. Shore 1988	"The JNIDS Development Cycle." in *Proceedings, Second International Software for Strategic Systems Conference.* October 1988, Huntsville, Alabama.
JNIDS 1993	Partnerships. *JNIDS Newsletter* 3,1.
Luftman, S., S. Lewis, and S. Oldach 1993	Transforming the Enterprise: the Alignment of Business and Information Strategies. *IBM Systems Journal.* March 1993.
Martin, C. 1988	*User-Centered Requirements Analysis.* Englewood Cliffs, New Jersey: Prentice Hall.
Martin, J. 1991	*Rapid Application Development.* MacMillian.
Martin, R. 1994	The Moving Wave of Technology. *Journal of Systems Management.* February 1994.
Moad, J. 1994	Welcome to the Virtual IS Organization. *Datamation.* February 1, 1994.
National Information Display Lab 1992	*Softcopy the Newsletter of the National Information Display Lab,* David Sarnoff Research Center 2,2.
Norman, D., and S. Draper 1986	*User-Centered System Design New Perspectives on Human Computer Interaction.* Hillsdale, New Jersey: Lawrence Erlbaum Associates, Inc.
Parnas, D. 1979	Designing Software for Ease of Extension and Contraction. *IEEE Transactions on Software Engineering* SE-5,2.

Whiteside, J., J. Bennett, and K. Holtzblatt 1988 — "Usability Engineering: Our Experience and Evolution." In *Handbook of Human Computer Interaction*. Edited by M. Helander. North Holland, New York.

Wood, M., and I. Sommerville 1988 — "A Knowledge-Based Software Components Catalogue." In *Software Engineering Environments*. Edited by P. Brereton. John Wiley & Sons.

Wood, J., and D. Silver 1989 — *Joint Application Design*. New York, New York: Wiley and Sons.

Yin 1989 — *Case Study Research, Design and Methods*. Newbury Park, California: SAGE Publications, Inc.

This page intentionally left blank.

BIBLIOGRAPHY

Campbell, D., and J. Stanley. *Experimental and Quasi-Experimental Designs for Research*. Chicago, Illinois: Rand McNally, 1966.

Clifton, T., and T. Starr. "Application of Software Engineering Estimation and Measurement Techniques to Rapid Prototyping/Iterative Development Projects." In Structure Development Forum XIII. 1993.

Comaford, C. "Timeboxing Let's You Whistle While You Work." *PC Week*. February 7, 1994.

Couger, D. L. "End User Computing" *Harvard Business Review*. September–October 1986.

Curtis, B., H. Krasner, and N. Iscoe. "A Field Study of the Software Design Process for Large Systems." *Communications of the ACM*. November 1988.

Dickey, S. "Rolling ith the Punches: In Rightsizing, How Do Users Know What Systems Are Right for Them?" *Midrange Systems*. November 9, 1993.

Ehn, P. "Scandinavian Design: On Participation and Skill" in *Usability Turning Technologies Into Tools*. New York, New York: Oxford University Press, 1992.

Frankl, V. *Man's Search for Meaning*. New York, New York: Washington Square Press, 1959.

Gould, J., and C. Lewis. *Designing for Usability: Key Principles and What Designers Think*, 1985.

Hollander, N., and N. Mirlocca. "Working Out Problems in Workshops; Bring Users Into the Loop Before You Install New Technology." *Information Week*. January 31, 1994.

Hough, D. "An Evolutionary Approach for Application Development." *IBM Systems Journal*. September 19, 1993.

Kouzes, J., and B. Posner. *The Leadership Challenge How to Get Extraordinary Things Done in Organizations*. San Francisco, California: Jossey-Bass Publishers, 1987.

Kukla, C., E. Clemens, R. Morse, and D. Cash. "Designing Effective Systems: A Tool Approach." In *Usability Turning Technologies Into Tools*. New York, New York: Oxford University Press, 1992.

Louderback, J. "Client/server: It's Neither a Dessert Wax Nor a Floor Topping." *PC Week*. April 25, 1994.

Madsen, K., and P. Aiken. "Experiences Using Cooperative Interactive Storyboard Prototyping." *Communications of the ACM*. June 1993.

Martin, J. *Design and Strategy for Distributed Data Processing*. New Jersey: Prentice Hall, 1981.

Matsubara, T. "Bringing Up Software Designers." *American Programmers*. July/August 1990.

McComb, M. "CASE Tools Implementation at Amtrak—Lessons Almost Learned." *Journal of Systems Management*. March 1994.

Musen, M. *Automated Generation of Model-Based Knowledge-Acquisition Tools*. San Mateo, California: Morgan Kaufmann Publishers, Inc., 1989.

Oresky, D., and C. Haapala. "Iterative Software Development: Achieving Quality Through Verification and Validation." In *Proceedings of the 6th International Software Quality Week '93*, May, 1993, San Francisco, California.

Rheinfrank, J., W. Hartman, and A. Wasserman. "Design for Usability: Crafting a Strategy for the Design of a New Generation of Xerox Copiers." In *Usability Turning Technologies Into Tools*. New York, New York: Oxford University Press, 1992.

Von Hippel, E. *The Sources of Innovation*. Oxford University Press, 1988.

Williams, M., and V. Begg. "Translation Between Software Designers and Users." *CACM*. June 1993.

Winant, R. "The Measure of Success." *The Computer Conference Analysis Newsletter*. September 9, 1993.